Constraint Solving

René Thiemann and Fabian Mitterwallner

based on a previous course by Aart Middeldorp

Motivation for extending SAT to first-order theories

predicates instead of propositional variables

Motivation for extending SAT to first-order theories

predicates instead of propositional variables

Examples

- equalities and disequalities over the reals

$$(x_1 = x_2 \vee x_1 = x_3) \wedge (x_1 = x_2 \vee x_1 = x_4) \wedge x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge x_1 \neq x_4$$

Motivation for extending SAT to first-order theories

predicates instead of propositional variables

Examples

- equalities and disequalities over the reals

$$(x_1 = x_2 \vee x_1 = x_3) \wedge (x_1 = x_2 \vee x_1 = x_4) \wedge x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge x_1 \neq x_4$$

- boolean combination of linear-arithmethic predicates

$$(x_1 + 2x_3 < 5) \vee \neg(x_3 \leqslant 1) \wedge (x_1 \geqslant x_3)$$

Motivation for extending SAT to first-order theories

predicates instead of propositional variables

Examples

- equalities and disequalities over the reals

$$(x_1 = x_2 \vee x_1 = x_3) \wedge (x_1 = x_2 \vee x_1 = x_4) \wedge x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge x_1 \neq x_4$$

- boolean combination of linear-arithmethic predicates

$$(x_1 + 2x_3 < 5) \vee \neg(x_3 \leqslant 1) \wedge (x_1 \geqslant x_3)$$

- formula over arrays

$$(i = j \wedge a[j] = 1) \wedge \neg(a[i] = 1)$$

Outline

Definitions (Syntax)

- terms are built from function symbols and variables according to following BNF grammar:

$$t \ ::= \ x \mid f(t, \ldots, t)$$

Definitions (Syntax)

- terms are built from function symbols and variables according to following BNF grammar:

$$t \ ::= \ x \mid f(t,\ldots,t)$$

- formulas are built from predicate symbols, terms, connectives, and quantifiers according to following BNF grammar:

$$\varphi \ ::= \ P \mid P(t,\ldots,t) \mid \bot \mid \top \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \to \varphi) \mid (\forall x.\,\varphi) \mid (\exists x.\,\varphi)$$

Definitions (Syntax)

- terms are built from function symbols and variables according to following BNF grammar:

$$t \ ::= \ x \mid f(t, \dots, t)$$

- formulas are built from predicate symbols, terms, connectives, and quantifiers according to following BNF grammar:

$$\varphi \ ::= \ P \mid P(t, \dots, t) \mid \bot \mid \top \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \rightarrow \varphi) \mid (\forall x.\, \varphi) \mid (\exists x.\, \varphi)$$

- notational conventions:
  - binding precedence $\neg \quad > \quad \wedge \quad > \quad \vee \quad > \quad \rightarrow \quad > \quad \forall, \exists$
  - omit outer parentheses, compress quantifiers: $\forall x\, y.\, \varphi$ instead of $\forall x. \forall y. \varphi$
  - $\rightarrow$, $\wedge$, $\vee$ are right-associative
  - constants c are written without parentheses: c instead of $c()$

Definitions (Syntax)

- terms are built from function symbols and variables according to following BNF grammar:

$$t \ ::= \ x \mid f(t, \dots, t)$$

- formulas are built from predicate symbols, terms, connectives, and quantifiers according to following BNF grammar:

$$\varphi \ ::= \ P \mid P(t, \dots, t) \mid \bot \mid \top \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \to \varphi) \mid (\forall x.\, \varphi) \mid (\exists x.\, \varphi)$$

- notational conventions:
  - binding precedence $\neg \ > \ \wedge \ > \ \vee \ > \ \to \ > \ \forall, \exists$
  - omit outer parentheses, compress quantifiers: $\forall x\, y.\, \varphi$ instead of $\forall x. \forall y. \varphi$
  - $\to$, $\wedge$, $\vee$ are right-associative
  - constants c are written without parentheses: c instead of $c()$
- sentence is formula without free variables

Definitions (Semantics)

- model $\mathcal{M}$ for pair $(\mathcal{F}, \mathcal{P})$ with function (predicate) symbols $\mathcal{F}$ ($\mathcal{P}$) consists of
  1. non-empty set A

Definitions (Semantics)

- model $\mathcal{M}$ for pair $(\mathcal{F}, \mathcal{P})$ with function (predicate) symbols $\mathcal{F}$ ($\mathcal{P}$) consists of

  1. non-empty set A
  2. function $f^{\mathcal{M}} \colon A^n \to A$ for every n-ary function symbol $f \in \mathcal{F}$

Definitions (Semantics)

- model $\mathcal{M}$ for pair $(\mathcal{F}, \mathcal{P})$ with function (predicate) symbols $\mathcal{F}$ ($\mathcal{P}$) consists of
  1. non-empty set A
  2. function $f^{\mathcal{M}}\colon A^n \to A$ for every n-ary function symbol $f \in \mathcal{F}$
  3. subset $P^{\mathcal{M}} \subseteq A^n$ for every n-ary predicate symbol $P \in \mathcal{P}$

Definitions (Semantics)

- model $\mathcal{M}$ for pair $(\mathcal{F}, \mathcal{P})$ with function (predicate) symbols $\mathcal{F}$ ($\mathcal{P}$) consists of
  1. non-empty set A
  2. function $f^{\mathcal{M}} \colon A^n \to A$ for every n-ary function symbol $f \in \mathcal{F}$
  3. subset $P^{\mathcal{M}} \subseteq A^n$ for every n-ary predicate symbol $P \in \mathcal{P}$
- environment for model $\mathcal{M} = (A, \{f^{\mathcal{M}}\}_{f \in \mathcal{F}}, \{P^{\mathcal{M}}\}_{P \in \mathcal{P}})$ is mapping l from variables to A

Definitions (Semantics)

- model $\mathcal{M}$ for pair $(\mathcal{F}, \mathcal{P})$ with function (predicate) symbols $\mathcal{F}$ ($\mathcal{P}$) consists of
  1. non-empty set A
  2. function $f^{\mathcal{M}} \colon A^n \to A$ for every n-ary function symbol $f \in \mathcal{F}$
  3. subset $P^{\mathcal{M}} \subseteq A^n$ for every n-ary predicate symbol $P \in \mathcal{P}$
- environment for model $\mathcal{M} = (A, \{f^{\mathcal{M}}\}_{f \in \mathcal{F}}, \{P^{\mathcal{M}}\}_{P \in \mathcal{P}})$ is mapping l from variables to A
- value $t^{\mathcal{M},l}$ of term t in model $\mathcal{M}$ relative to environment l is defined inductively:

$$t^{\mathcal{M},l} = \begin{cases} l(t) & \text{if } t \text{ is variable} \\ f^{\mathcal{M}}(t_1^{\mathcal{M},l}, \dots, t_n^{\mathcal{M},l}) & \text{if } t = f(t_1, \dots, t_n) \end{cases}$$

Definitions (Semantics)

- model $\mathcal{M}$ for pair $(\mathcal{F}, \mathcal{P})$ with function (predicate) symbols $\mathcal{F}$ $(\mathcal{P})$ consists of
  1. non-empty set A
  2. function $f^{\mathcal{M}} \colon A^n \to A$ for every n-ary function symbol $f \in \mathcal{F}$
  3. subset $P^{\mathcal{M}} \subseteq A^n$ for every n-ary predicate symbol $P \in \mathcal{P}$
- environment for model $\mathcal{M} = (A, \{f^{\mathcal{M}}\}_{f \in \mathcal{F}}, \{P^{\mathcal{M}}\}_{P \in \mathcal{P}})$ is mapping I from variables to A
- value $t^{\mathcal{M},I}$ of term t in model $\mathcal{M}$ relative to environment I is defined inductively:

$$t^{\mathcal{M},I} = \begin{cases} I(t) & \text{if } t \text{ is variable} \\ f^{\mathcal{M}}(t_1^{\mathcal{M},I}, \dots, t_n^{\mathcal{M},I}) & \text{if } t = f(t_1, \dots, t_n) \end{cases}$$

- given environment I, variable x and element $a \in A$, environment $I[x \mapsto a]$ is defined as

$$(I[x \mapsto a])(y) = \begin{cases} a & \text{if } y = x \\ I(y) & \text{if } y \neq x \end{cases}$$

Definition (Semantics)

satisfaction relation $\mathcal{M} \vDash_I \varphi$ is defined by induction on structure of φ

Definition (Semantics)

satisfaction relation $\mathcal{M} \vDash_I \varphi$ is defined by induction on structure of φ:

$$\mathcal{M} \vDash_I \varphi \iff \begin{cases} (t_1^{\mathcal{M},I}, \dots, t_n^{\mathcal{M},I}) \in P^{\mathcal{M}} & \text{if } \varphi = P(t_1, \dots, t_n) \end{cases}$$

Definition (Semantics)

satisfaction relation $\mathcal{M} \vDash_I \varphi$ is defined by induction on structure of φ:

$$\mathcal{M} \vDash_I \varphi \iff \begin{cases} (t_1^{\mathcal{M},I}, \dots, t_n^{\mathcal{M},I}) \in P^{\mathcal{M}} & \text{if } \varphi = P(t_1, \dots, t_n) \\ \mathcal{M} \nvDash_I \psi & \text{if } \varphi = \neg\psi \end{cases}$$

Definition (Semantics)

satisfaction relation $\mathcal{M} \vDash_I \varphi$ is defined by induction on structure of φ:

$$\mathcal{M} \vDash_I \varphi \iff \begin{cases} (t_1^{\mathcal{M},I}, \ldots, t_n^{\mathcal{M},I}) \in P^{\mathcal{M}} & \text{if } \varphi = P(t_1, \ldots, t_n) \\ \mathcal{M} \nvDash_I \psi & \text{if } \varphi = \neg\psi \end{cases}$$

Notation

$\mathcal{M} \nvDash_I \psi$ denotes "not $\mathcal{M} \vDash_I \psi$"

Definition (Semantics)

satisfaction relation $\mathcal{M} \vDash_I \varphi$ is defined by induction on structure of φ:

$$\mathcal{M} \vDash_I \varphi \iff \begin{cases} (t_1^{\mathcal{M},I}, \ldots, t_n^{\mathcal{M},I}) \in P^{\mathcal{M}} & \text{if } \varphi = P(t_1, \ldots, t_n) \\ \mathcal{M} \nvDash_I \psi & \text{if } \varphi = \neg\psi \\ \mathcal{M} \vDash_I \psi_1 \text{ and } \mathcal{M} \vDash_I \psi_2 & \text{if } \varphi = (\psi_1 \wedge \psi_2) \end{cases}$$

Notation

$\mathcal{M} \nvDash_I \psi$ denotes "not $\mathcal{M} \vDash_I \psi$"

Definition (Semantics)

satisfaction relation $\mathcal{M} \vDash_I \varphi$ is defined by induction on structure of φ:

$$\mathcal{M} \vDash_I \varphi \iff \begin{cases} (t_1^{\mathcal{M},I}, \ldots, t_n^{\mathcal{M},I}) \in P^{\mathcal{M}} & \text{if } \varphi = P(t_1, \ldots, t_n) \\ \mathcal{M} \nvDash_I \psi & \text{if } \varphi = \neg\psi \\ \mathcal{M} \vDash_I \psi_1 \text{ and } \mathcal{M} \vDash_I \psi_2 & \text{if } \varphi = (\psi_1 \wedge \psi_2) \\ \mathcal{M} \vDash_I \psi_1 \text{ or } \mathcal{M} \vDash_I \psi_2 & \text{if } \varphi = (\psi_1 \vee \psi_2) \end{cases}$$

Notation

$\mathcal{M} \nvDash_I \psi$ denotes "not $\mathcal{M} \vDash_I \psi$"

Definition (Semantics)

satisfaction relation $\mathcal{M} \vDash_I \varphi$ is defined by induction on structure of φ:

$$\mathcal{M} \vDash_I \varphi \iff \begin{cases} (t_1^{\mathcal{M},I}, \ldots, t_n^{\mathcal{M},I}) \in P^{\mathcal{M}} & \text{if } \varphi = P(t_1, \ldots, t_n) \\ \mathcal{M} \nvDash_I \psi & \text{if } \varphi = \neg\psi \\ \mathcal{M} \vDash_I \psi_1 \text{ and } \mathcal{M} \vDash_I \psi_2 & \text{if } \varphi = (\psi_1 \wedge \psi_2) \\ \mathcal{M} \vDash_I \psi_1 \text{ or } \mathcal{M} \vDash_I \psi_2 & \text{if } \varphi = (\psi_1 \vee \psi_2) \\ \mathcal{M} \nvDash_I \psi_1 \text{ or } \mathcal{M} \vDash_I \psi_2 & \text{if } \varphi = (\psi_1 \rightarrow \psi_2) \end{cases}$$

Notation

$\mathcal{M} \nvDash_I \psi$ denotes "not $\mathcal{M} \vDash_I \psi$"

Definition (Semantics)

satisfaction relation $\mathcal{M} \vDash_I \varphi$ is defined by induction on structure of φ:

$$
\mathcal{M} \vDash_I \varphi \iff \begin{cases}
(t_1^{\mathcal{M},I}, \dots, t_n^{\mathcal{M},I}) \in P^{\mathcal{M}} & \text{if } \varphi = P(t_1, \dots, t_n) \\
\mathcal{M} \nvDash_I \psi & \text{if } \varphi = \neg\psi \\
\mathcal{M} \vDash_I \psi_1 \text{ and } \mathcal{M} \vDash_I \psi_2 & \text{if } \varphi = (\psi_1 \wedge \psi_2) \\
\mathcal{M} \vDash_I \psi_1 \text{ or } \mathcal{M} \vDash_I \psi_2 & \text{if } \varphi = (\psi_1 \vee \psi_2) \\
\mathcal{M} \nvDash_I \psi_1 \text{ or } \mathcal{M} \vDash_I \psi_2 & \text{if } \varphi = (\psi_1 \to \psi_2) \\
\mathcal{M} \vDash_{I[x \mapsto a]} \psi \text{ for all } a \in A & \text{if } \varphi = \forall x.\, \psi
\end{cases}
$$

Notation

$\mathcal{M} \nvDash_I \psi$ denotes "not $\mathcal{M} \vDash_I \psi$"

Definition (Semantics)

satisfaction relation $\mathcal{M} \vDash_I \varphi$ is defined by induction on structure of φ:

$$\mathcal{M} \vDash_I \varphi \iff \begin{cases} (t_1^{\mathcal{M},I}, \dots, t_n^{\mathcal{M},I}) \in P^{\mathcal{M}} & \text{if } \varphi = P(t_1, \dots, t_n) \\ \mathcal{M} \nvDash_I \psi & \text{if } \varphi = \neg\psi \\ \mathcal{M} \vDash_I \psi_1 \text{ and } \mathcal{M} \vDash_I \psi_2 & \text{if } \varphi = (\psi_1 \wedge \psi_2) \\ \mathcal{M} \vDash_I \psi_1 \text{ or } \mathcal{M} \vDash_I \psi_2 & \text{if } \varphi = (\psi_1 \vee \psi_2) \\ \mathcal{M} \nvDash_I \psi_1 \text{ or } \mathcal{M} \vDash_I \psi_2 & \text{if } \varphi = (\psi_1 \rightarrow \psi_2) \\ \mathcal{M} \vDash_{I[x \mapsto a]} \psi \text{ for all } a \in A & \text{if } \varphi = \forall x.\, \psi \\ \mathcal{M} \vDash_{I[x \mapsto a]} \psi \text{ for some } a \in A & \text{if } \varphi = \exists x.\, \psi \end{cases}$$

Notation

$\mathcal{M} \nvDash_I \psi$ denotes "not $\mathcal{M} \vDash_I \psi$"

Definition (Semantics)

satisfaction relation $\mathcal{M} \vDash_I \varphi$ is defined by induction on structure of φ:

$$\mathcal{M} \vDash_I \top$$

$$\mathcal{M} \nvDash_I \bot$$

$$\mathcal{M} \vDash_I \varphi \iff \begin{cases} (t_1^{\mathcal{M},I}, \dots, t_n^{\mathcal{M},I}) \in P^{\mathcal{M}} & \text{if } \varphi = P(t_1, \dots, t_n) \\ \mathcal{M} \nvDash_I \psi & \text{if } \varphi = \neg\psi \\ \mathcal{M} \vDash_I \psi_1 \text{ and } \mathcal{M} \vDash_I \psi_2 & \text{if } \varphi = (\psi_1 \wedge \psi_2) \\ \mathcal{M} \vDash_I \psi_1 \text{ or } \mathcal{M} \vDash_I \psi_2 & \text{if } \varphi = (\psi_1 \vee \psi_2) \\ \mathcal{M} \nvDash_I \psi_1 \text{ or } \mathcal{M} \vDash_I \psi_2 & \text{if } \varphi = (\psi_1 \rightarrow \psi_2) \\ \mathcal{M} \vDash_{I[x \mapsto a]} \psi \text{ for all } a \in A & \text{if } \varphi = \forall x.\, \psi \\ \mathcal{M} \vDash_{I[x \mapsto a]} \psi \text{ for some } a \in A & \text{if } \varphi = \exists x.\, \psi \end{cases}$$

Notation

$\mathcal{M} \nvDash_I \psi$ denotes "not $\mathcal{M} \vDash_I \psi$"

Definitions

formula ψ

- ψ is satisfiable if $\mathcal{M} \vDash_I \psi$ for some model $\mathcal{M}$ and environment I

Definitions

formula ψ and (possibly infinite) set of formulas Γ

- ψ is satisfiable if $\mathcal{M} \vDash_I \psi$ for some model $\mathcal{M}$ and environment I
- Γ is satisfiable (consistent) if $\mathcal{M} \vDash_I \varphi$ for all $\varphi \in \Gamma$, for some model $\mathcal{M}$ and environment I

Definitions

formula ψ and (possibly infinite) set of formulas Γ

- ψ is satisfiable if $\mathcal{M} \vDash_I \psi$ for some model $\mathcal{M}$ and environment I
- Γ is satisfiable (consistent) if $\mathcal{M} \vDash_I \varphi$ for all $\varphi \in \Gamma$, for some model $\mathcal{M}$ and environment I
- ψ is valid if $\mathcal{M} \vDash_I \psi$ for all (appropriate) models $\mathcal{M}$ and environments I

Definitions

formula ψ and (possibly infinite) set of formulas Γ

- ψ is satisfiable if $\mathcal{M} \vDash_I \psi$ for some model $\mathcal{M}$ and environment I
- Γ is satisfiable (consistent) if $\mathcal{M} \vDash_I \varphi$ for all $\varphi \in \Gamma$, for some model $\mathcal{M}$ and environment I
- ψ is valid if $\mathcal{M} \vDash_I \psi$ for all (appropriate) models $\mathcal{M}$ and environments I
- $\Gamma \vDash \psi$ (semantic entailment) if $\mathcal{M} \vDash_I \psi$ whenever $\mathcal{M} \vDash_I \varphi$ for all $\varphi \in \Gamma$, for all (appropriate) models $\mathcal{M}$ and environments I

Outline

First-Order Theories

formalize particular structures to enable reasoning about them

First-Order Theories

formalize particular structures to enable reasoning about them

Definition

first-order theory $T = (\Sigma, \mathcal{A})$ consists of

- signature Σ specifying function and predicate symbols

First-Order Theories

formalize particular structures to enable reasoning about them

Definition

first-order theory $T = (\Sigma, \mathcal{A})$ consists of

- signature Σ specifying function and predicate symbols
- axioms $\mathcal{A}$: sentences involving only function and predicate symbols from Σ

First-Order Theories

formalize particular structures to enable reasoning about them

Definition

first-order theory $T = (\Sigma, \mathcal{A})$ consists of

- signature Σ specifying function and predicate symbols
- axioms $\mathcal{A}$: sentences involving only function and predicate symbols from Σ

Remark

axioms $\mathcal{A}$ provide meaning to symbols of Σ

Example (Addition on Natural Numbers: Presburger Arithmetic)

- signature: constant 0 unary function symbol s binary symbols $=$ $+$
- axioms (in addition to axioms for equality)
  - $\forall x.\, s(x) \neq 0$
  - $\forall x\, y.\, s(x) = s(y) \to x = y$

Example (Addition on Natural Numbers: Presburger Arithmetic)

- signature: constant 0 unary function symbol s binary symbols $=$ $+$
- axioms (in addition to axioms for equality)
  - $\forall x.\, s(x) \neq 0$
  - $\forall x\, y.\, s(x) = s(y) \rightarrow x = y$
  - $\forall x.\, x + 0 = x$
  - $\forall x\, y.\, x + s(y) = s(x + y)$

Example (Addition on Natural Numbers: Presburger Arithmetic)

- signature: constant 0 unary function symbol s binary symbols $=$ $+$
- axioms (in addition to axioms for equality)
  - $\forall x.\, s(x) \neq 0$
  - $\forall x\, y.\, s(x) = s(y) \rightarrow x = y$
  - $\forall x.\, x + 0 = x$
  - $\forall x\, y.\, x + s(y) = s(x + y)$
  - induction

$$\psi(0) \wedge (\forall x.\, \psi(x) \rightarrow \psi(s(x))) \rightarrow \forall y.\, \psi(y)$$

    for every formula $\psi(x)$ with single free variable x

Example (Addition on Natural Numbers: Presburger Arithmetic)

- signature: constant 0 unary function symbol s binary symbols $=$ $+$
- axioms (in addition to axioms for equality)
  - $\forall x.\, s(x) \neq 0$
  - $\forall x\, y.\, s(x) = s(y) \rightarrow x = y$
  - $\forall x.\, x + 0 = x$
  - $\forall x\, y.\, x + s(y) = s(x + y)$
  - induction

$$\psi(0) \wedge (\forall x.\, \psi(x) \rightarrow \psi(s(x))) \rightarrow \forall y.\, \psi(y)$$

for every formula $\psi(x)$ with single free variable x

Remark

$>$ can be encoded: $x > y \iff \exists z.\, z \neq 0 \wedge x = y + z$

Example (Addition and Multiplication: Peano Arithmetic)

- signature: constant 0 unary function symbol s binary symbols $=$ $+$ $\times$
- axioms (PA)
  - $\forall x.\, s(x) \neq 0$
  - $\forall x\, y.\, s(x) = s(y) \rightarrow x = y$
  - $\forall x.\, x + 0 = x$
  - $\forall x\, y.\, x + s(y) = s(x + y)$
  - induction

    $$\psi(0) \wedge (\forall x.\, \psi(x) \rightarrow \psi(s(x))) \rightarrow \forall y.\, \psi(y)$$

    for every formula $\psi(x)$ with single free variable x
  - $\forall x.\, x \times 0 = 0$
  - $\forall x\, y.\, x \times s(y) = (x \times y) + x$

Definition

sentence ψ over Σ is valid in first-order theory $T = (\Sigma, \mathcal{A})$ if $\mathcal{M} \vDash \psi$ for every model $\mathcal{M}$ such that $\mathcal{M} \vDash \mathcal{A}$

Definition

sentence ψ over Σ is valid in first-order theory $T = (\Sigma, \mathcal{A})$ if $\mathcal{M} \vDash \psi$ for every model $\mathcal{M}$ such that $\mathcal{M} \vDash \mathcal{A}$ (notation: $T \vDash \psi$)

Definition

sentence ψ over Σ is valid in first-order theory $T = (\Sigma, \mathcal{A})$ if $\mathcal{M} \vDash \psi$ for every model $\mathcal{M}$ such that $\mathcal{M} \vDash \mathcal{A}$ (notation: $T \vDash \psi$)

Definitions

first-order theory $T = (\Sigma, \mathcal{A})$ is

- consistent (satisfiable) if $\mathcal{M} \vDash \mathcal{A}$ for some model $\mathcal{M}$

Definition

sentence ψ over Σ is valid in first-order theory $T = (\Sigma, \mathcal{A})$ if $\mathcal{M} \vDash \psi$ for every model $\mathcal{M}$ such that $\mathcal{M} \vDash \mathcal{A}$ (notation: $T \vDash \psi$)

Definitions

first-order theory $T = (\Sigma, \mathcal{A})$ is

- consistent (satisfiable) if $\mathcal{M} \vDash \mathcal{A}$ for some model $\mathcal{M}$
- complete if $T \vDash \psi$ or $T \vDash \neg\psi$ for every sentence ψ over Σ

Definition

sentence ψ over Σ is valid in first-order theory $T = (\Sigma, \mathcal{A})$ if $\mathcal{M} \vDash \psi$ for every model $\mathcal{M}$ such that $\mathcal{M} \vDash \mathcal{A}$ (notation: $T \vDash \psi$)

Definitions

first-order theory $T = (\Sigma, \mathcal{A})$ is

- consistent (satisfiable) if $\mathcal{M} \vDash \mathcal{A}$ for some model $\mathcal{M}$
- complete if $T \vDash \psi$ or $T \vDash \neg\psi$ for every sentence ψ over Σ
- decidable if validity problem

  instance: sentence ψ over Σ

  question: $T \vDash \psi$?

  is decidable

Theorem (Presburger 1929)

Presburger arithmetic is decidable

Theorem (Presburger 1929)

Presburger arithmetic is decidable

Theorem (Church 1936)

Peano arithmetic is undecidable

Theorem (Presburger 1929)

Presburger arithmetic is decidable

Theorem (Church 1936)

Peano arithmetic is undecidable

Theorem

Presburger and Peano arithmetic are not finitely axiomatizable

Theorem (Presburger 1929)

Presburger arithmetic is decidable

Theorem (Church 1936)

Peano arithmetic is undecidable

Theorem

Presburger and Peano arithmetic are not finitely axiomatizable

Definition

$\mathcal{N}$ denotes standard model of arithmetic:

- universe: $\mathbb{N}$
- $0^{\mathcal{N}} = 0 \quad s^{\mathcal{N}}(x) = x + 1 \quad +^{\mathcal{N}}(x, y) = x + y \quad \times^{\mathcal{N}}(x, y) = x \times y$

Theorem (Presburger 1929)

Presburger arithmetic is decidable

Theorem (Church 1936)

Peano arithmetic is undecidable

Theorem

Presburger and Peano arithmetic are not finitely axiomatizable

Definition

$\mathcal{N}$ denotes standard model of arithmetic:

- universe: $\mathbb{N}$
- $0^{\mathcal{N}} = 0 \quad s^{\mathcal{N}}(x) = x + 1 \quad +^{\mathcal{N}}(x,y) = x + y \quad \times^{\mathcal{N}}(x,y) = x \times y \quad (=^{\mathcal{N}} = \{(x,x) \mid x \in \mathbb{N}\})$

Theorem

$\mathcal{N} \models \mathsf{PA}$ (so Peano arithmetic is consistent)

Theorem

$\mathcal{N} \models \mathsf{PA}$ (so Peano arithmetic is consistent)

Gödel's Incompleteness Theorem

$\exists$ sentence ψ such that $\mathcal{N} \models \psi$ and $\mathsf{PA} \nvdash \psi$

Theorem

$\mathcal{N} \models \mathsf{PA}$ (so Peano arithmetic is consistent)

Gödel's Incompleteness Theorem

$\exists$ sentence ψ such that $\mathcal{N} \models \psi$ and $\mathsf{PA} \nvdash \psi$

Kurt Gödel 

Theorem

$\mathcal{N} \models \mathsf{PA}$ (so Peano arithmetic is consistent)

Gödel's Incompleteness Theorem

$\exists$ sentence ψ such that $\mathcal{N} \models \psi$ and $\mathsf{PA} \nvdash \psi$

Kurt Gödel

Proof Idea

sentence ψ encodes that ψ itself is unprovable in PA

Outline

Definition

fragment of theory $T = (\Sigma, \mathcal{A})$ is syntactically restricted subset of formulas over Σ

Definition

fragment of theory $T = (\Sigma, \mathcal{A})$ is syntactically restricted subset of formulas over Σ

- quantifier-free fragment: no quantifiers

Definition

fragment of theory $T = (\Sigma, \mathcal{A})$ is syntactically restricted subset of formulas over Σ

- quantifier-free fragment: no quantifiers
- conjunctive fragment: conjunction as only logical connective

Definition

fragment of theory $T = (\Sigma, \mathcal{A})$ is syntactically restricted subset of formulas over Σ

- quantifier-free fragment: no quantifiers
- conjunctive fragment: conjunction as only logical connective

Satisfiability Modulo Theories (SMT)

theories are identified with their standard model

Definition

fragment of theory $T = (\Sigma, \mathcal{A})$ is syntactically restricted subset of formulas over Σ

- quantifier-free fragment: no quantifiers
- conjunctive fragment: conjunction as only logical connective

Satisfiability Modulo Theories (SMT)

theories are identified with their standard model:

- domain is given explicitly

Definition

fragment of theory $T = (\Sigma, \mathcal{A})$ is syntactically restricted subset of formulas over Σ

- quantifier-free fragment: no quantifiers
- conjunctive fragment: conjunction as only logical connective

Satisfiability Modulo Theories (SMT)

theories are identified with their standard model:

- domain is given explicitly
- interpretation of symbols is in accordance with their common use

Definition

fragment of theory $T = (\Sigma, \mathcal{A})$ is syntactically restricted subset of formulas over Σ

- quantifier-free fragment: no quantifiers
- conjunctive fragment: conjunction as only logical connective

Satisfiability Modulo Theories (SMT)

theories are identified with their standard model:

- domain is given explicitly
- interpretation of symbols is in accordance with their common use
- formulas are often restricted to quantifier-free fragment

Example (Binairo)

The objective of the number placement puzzle binairo is to fill a grid with 0's and 1's, where there is an equal number of 0's and 1's and no more than two consecutive 0's or 1's in each row and column. Additionally, identical rows and identical columns are forbidden. For instance, the binairo puzzle on the left has the solution on the right:

	0		0		0
	1			0	
0		1			
					1
		1		0	
				0	

1	0	1	0	1	0
1	1	0	1	0	0
0	0	1	0	1	1
0	1	0	0	1	1
1	0	1	1	0	0
0	1	0	1	0	1

Example (Binairo)

The objective of the number placement puzzle binairo is to fill a grid with 0’s and 1’s, where there is an equal number of 0’s and 1’s and no more than two consecutive 0’s or 1’s in each row and column. Additionally, identical rows and identical columns are forbidden. For instance, the binairo puzzle on the left has the solution on the right:

	0		0		0
	1			0	
0		1			
					1
		1		0	
				0	

1					
			0		
					1
	0				
1		1			1

1	0	1	0	1	0
1	1	0	1	0	0
0	0	1	0	1	1
0	1	0	0	1	1
1	0	1	1	0	0
0	1	0	1	0	1

Example (Binairo)

The objective of the number placement puzzle binairo is to fill a grid with 0's and 1's, where there is an equal number of 0's and 1's and no more than two consecutive 0's or 1's in each row and column. Additionally, identical rows and identical columns are forbidden. For instance, the binairo puzzle on the left has the solution on the right:

	0		0		0
	1			0	
0		1			
					1
		1		0	
				0	

1	2,6	3,6	4,6	5,6	6,6
1,5	2,5	3,5	0	5,5	6,5
1,4	2,4	3,4	4,4	5,4	1
1,3	2,3	3,3	4,3	5,3	6,3
1,2	0	3,2	4,2	5,2	6,2
1	2,1	1	4,1	5,1	1

1	0	1	0	1	0
1	1	0	1	0	0
0	0	1	0	1	1
0	1	0	0	1	1
1	0	1	1	0	0
0	1	0	1	0	1

Remark

- SAT CNF encoding is tedious

Remark

- SAT CNF encoding is tedious

SMT Encoding

$$\bigwedge_{i=1}^{6} \bigwedge_{j=1}^{6} \left(x_{i,j} = 0 \vee x_{i,j} = 1\right)$$

Remark

- SAT CNF encoding is tedious

SMT Encoding

$$\bigwedge_{i=1}^{6} \bigwedge_{j=1}^{6} \left(x_{i,j} = 0 \vee x_{i,j} = 1\right) \ \wedge \ \bigwedge_{i=1}^{6} \left(\sum_{j=1}^{6} x_{i,j} = 3\right) \ \wedge \ \bigwedge_{j=1}^{6} \left(\sum_{i=1}^{6} x_{i,j} = 3\right)$$

Remark

- SAT CNF encoding is tedious

SMT Encoding

$$\bigwedge_{i=1}^{6} \bigwedge_{j=1}^{6} (x_{i,j} = 0 \vee x_{i,j} = 1) \;\wedge\; \bigwedge_{i=1}^{6} \left(\sum_{j=1}^{6} x_{i,j} = 3 \right) \;\wedge\; \bigwedge_{j=1}^{6} \left(\sum_{i=1}^{6} x_{i,j} = 3 \right) \;\wedge$$

$$\bigwedge_{i=1}^{4} \bigwedge_{j=1}^{6} \left(\sum_{k=0}^{2} x_{i+k,j} = 1 \vee \sum_{k=0}^{2} x_{i+k,j} = 2 \right)$$

Remark

- SAT CNF encoding is tedious

SMT Encoding

$$\bigwedge_{i=1}^{6} \bigwedge_{j=1}^{6} (x_{i,j} = 0 \vee x_{i,j} = 1) \;\wedge\; \bigwedge_{i=1}^{6} \left(\sum_{j=1}^{6} x_{i,j} = 3 \right) \;\wedge\; \bigwedge_{j=1}^{6} \left(\sum_{i=1}^{6} x_{i,j} = 3 \right) \;\wedge$$

$$\bigwedge_{i=1}^{4} \bigwedge_{j=1}^{6} \left(\sum_{k=0}^{2} x_{i+k,j} = 1 \vee \sum_{k=0}^{2} x_{i+k,j} = 2 \right) \;\wedge\; \bigwedge_{i=1}^{6} \bigwedge_{j=1}^{4} \left(\sum_{k=0}^{2} x_{i,j+k} = 1 \vee \sum_{k=0}^{2} x_{i,j+k} = 2 \right)$$

Remark

- SAT CNF encoding is tedious

SMT Encoding

$$\bigwedge_{i=1}^{6}\bigwedge_{j=1}^{6}(x_{i,j}=0 \vee x_{i,j}=1) \wedge \bigwedge_{i=1}^{6}\left(\sum_{j=1}^{6} x_{i,j}=3\right) \wedge \bigwedge_{j=1}^{6}\left(\sum_{i=1}^{6} x_{i,j}=3\right) \wedge$$

$$\bigwedge_{i=1}^{4}\bigwedge_{j=1}^{6}\left(\sum_{k=0}^{2} x_{i+k,j}=1 \vee \sum_{k=0}^{2} x_{i+k,j}=2\right) \wedge \bigwedge_{i=1}^{6}\bigwedge_{j=1}^{4}\left(\sum_{k=0}^{2} x_{i,j+k}=1 \vee \sum_{k=0}^{2} x_{i,j+k}=2\right) \wedge$$

$$\bigwedge_{i=1}^{5}\bigwedge_{k=i+1}^{6}\left(\bigvee_{j=1}^{6} x_{i,j}\neq x_{k,j}\right) \wedge \bigwedge_{j=1}^{5}\bigwedge_{k=j+1}^{6}\left(\bigvee_{i=1}^{6} x_{i,j}\neq x_{i,k}\right)$$

Remark

- SAT CNF encoding is tedious

SMT Encoding

$$\bigwedge_{i=1}^{6}\bigwedge_{j=1}^{6}(x_{i,j}=0 \vee x_{i,j}=1) \;\wedge\; \bigwedge_{i=1}^{6}\left(\sum_{j=1}^{6} x_{i,j}=3\right) \;\wedge\; \bigwedge_{j=1}^{6}\left(\sum_{i=1}^{6} x_{i,j}=3\right) \;\wedge$$

$$\bigwedge_{i=1}^{4}\bigwedge_{j=1}^{6}\left(\sum_{k=0}^{2} x_{i+k,j}=1 \vee \sum_{k=0}^{2} x_{i+k,j}=2\right) \;\wedge\; \bigwedge_{i=1}^{6}\bigwedge_{j=1}^{4}\left(\sum_{k=0}^{2} x_{i,j+k}=1 \vee \sum_{k=0}^{2} x_{i,j+k}=2\right) \;\wedge$$

$$\bigwedge_{i=1}^{5}\bigwedge_{k=i+1}^{6}\left(\bigvee_{j=1}^{6} x_{i,j}\neq x_{k,j}\right) \;\wedge\; \bigwedge_{j=1}^{5}\bigwedge_{k=j+1}^{6}\left(\bigvee_{i=1}^{6} x_{i,j}\neq x_{i,k}\right) \;\wedge$$

$$x_{2,2}=0 \;\wedge\; x_{4,5}=0 \;\wedge\; x_{1,1}=1 \;\wedge\; x_{3,1}=1 \;\wedge\; x_{6,1}=1 \;\wedge\; x_{6,4}=1 \;\wedge\; x_{1,6}=1$$

Remark

- SAT CNF encoding is tedious

SMT Encoding (Linear Integer Arithmetic)

$$\bigwedge_{i=1}^{6} \bigwedge_{j=1}^{6} (x_{i,j} = 0 \vee x_{i,j} = 1) \;\wedge\; \bigwedge_{i=1}^{6} \left(\sum_{j=1}^{6} x_{i,j} = 3 \right) \;\wedge\; \bigwedge_{j=1}^{6} \left(\sum_{i=1}^{6} x_{i,j} = 3 \right) \;\wedge$$

$$\bigwedge_{i=1}^{4} \bigwedge_{j=1}^{6} \left(\sum_{k=0}^{2} x_{i+k,j} = 1 \vee \sum_{k=0}^{2} x_{i+k,j} = 2 \right) \;\wedge\; \bigwedge_{i=1}^{6} \bigwedge_{j=1}^{4} \left(\sum_{k=0}^{2} x_{i,j+k} = 1 \vee \sum_{k=0}^{2} x_{i,j+k} = 2 \right) \;\wedge$$

$$\bigwedge_{i=1}^{5} \bigwedge_{k=i+1}^{6} \left(\bigvee_{j=1}^{6} x_{i,j} \neq x_{k,j} \right) \;\wedge\; \bigwedge_{j=1}^{5} \bigwedge_{k=j+1}^{6} \left(\bigvee_{i=1}^{6} x_{i,j} \neq x_{i,k} \right) \;\wedge$$

$$x_{2,2} = 0 \;\wedge\; x_{4,5} = 0 \;\wedge\; x_{1,1} = 1 \;\wedge\; x_{3,1} = 1 \;\wedge\; x_{6,1} = 1 \;\wedge\; x_{6,4} = 1 \;\wedge\; x_{1,6} = 1$$

SMT-LIB 2 Format

```
(declare-const x11 Int)  ...  (declare-const x66 Int)
```

SMT-LIB 2 Format

```
(declare-const x11 Int)  ...  (declare-const x66 Int)
(assert (or (= x11 0) (= x11 1)))  ...  (assert (or (= x66 0) (= x66 1)))
```

SMT-LIB 2 Format

```
(declare-const x11 Int)  ...  (declare-const x66 Int)
(assert (or (= x11 0) (= x11 1)))  ...  (assert (or (= x66 0) (= x66 1)))
(assert (= (+ x11 x12 x13 x14 x15 x16) 3))
...
(assert (= (+ x16 x26 x36 x46 x56 x66) 3))
```

SMT-LIB 2 Format

```
(declare-const x11 Int)  ...  (declare-const x66 Int)
(assert (or (= x11 0) (= x11 1)))  ...  (assert (or (= x66 0) (= x66 1)))
(assert (= (+ x11 x12 x13 x14 x15 x16) 3))
...
(assert (= (+ x16 x26 x36 x46 x56 x66) 3))
(assert (or (= (+ x11 x21 x31) 1) (= (+ x11 x21 x31) 2)))
...
(assert (or (= (+ x64 x65 x66) 1) (= (+ x64 x65 x66) 2)))
```

SMT-LIB 2 Format

```
(declare-const x11 Int)  ...  (declare-const x66 Int)
(assert (or (= x11 0) (= x11 1)))  ...  (assert (or (= x66 0) (= x66 1)))
(assert (= (+ x11 x12 x13 x14 x15 x16) 3))
...
(assert (= (+ x16 x26 x36 x46 x56 x66) 3))
(assert (or (= (+ x11 x21 x31) 1) (= (+ x11 x21 x31) 2)))
...
(assert (or (= (+ x64 x65 x66) 1) (= (+ x64 x65 x66) 2)))
(assert (or (not (= x11 x21)) (not (= x12 x22)) (not (= x13 x23))
            (not (= x14 x24)) (not (= x15 x25)) (not (= x16 x26))))
...
(assert (or (not (= x15 x16)) (not (= x25 x26)) (not (= x35 x36))
            (not (= x45 x46)) (not (= x55 x56)) (not (= x65 x66))))
(assert (= x22 0))   (assert (= x45 0))  ...  (assert (= x16 0))
```

SMT-LIB 2 Format

```
(declare-const x11 Int)  ...  (declare-const x66 Int)
(assert (or (= x11 0) (= x11 1)))  ...  (assert (or (= x66 0) (= x66 1)))
(assert (= (+ x11 x12 x13 x14 x15 x16) 3))
...
(assert (= (+ x16 x26 x36 x46 x56 x66) 3))
(assert (or (= (+ x11 x21 x31) 1) (= (+ x11 x21 x31) 2)))
...
(assert (or (= (+ x64 x65 x66) 1) (= (+ x64 x65 x66) 2)))
(assert (or (not (= x11 x21)) (not (= x12 x22)) (not (= x13 x23))
            (not (= x14 x24)) (not (= x15 x25)) (not (= x16 x26))))
...
(assert (or (not (= x15 x16)) (not (= x25 x26)) (not (= x35 x36))
            (not (= x45 x46)) (not (= x55 x56)) (not (= x65 x66))))
(assert (= x22 0))   (assert (= x45 0))  ...  (assert (= x16 0))
(check-sat)
```

SMT-LIB 2 Format

```
(declare-const x11 Int)  ...  (declare-const x66 Int)
(assert (or (= x11 0) (= x11 1)))  ...  (assert (or (= x66 0) (= x66 1)))
(assert (= (+ x11 x12 x13 x14 x15 x16) 3))
...
(assert (= (+ x16 x26 x36 x46 x56 x66) 3))
(assert (or (= (+ x11 x21 x31) 1) (= (+ x11 x21 x31) 2)))
...
(assert (or (= (+ x64 x65 x66) 1) (= (+ x64 x65 x66) 2)))
(assert (or (not (= x11 x21)) (not (= x12 x22)) (not (= x13 x23))
            (not (= x14 x24)) (not (= x15 x25)) (not (= x16 x26))))
...
(assert (or (not (= x15 x16)) (not (= x25 x26)) (not (= x35 x36))
            (not (= x45 x46)) (not (= x55 x56)) (not (= x65 x66))))
(assert (= x22 0))   (assert (= x45 0))  ...  (assert (= x16 0))
(check-sat)
(get-model)
```

```
(declare-const x11 Int)  ...  (declare-const x66 Int)
(assert (or (= x11 0) (= x11 1)))  ...  (assert (or (= x66 0) (= x66 1)))
(assert (= (+ x11 x12 x13 x14 x15 x16) 3))
...
(assert (= (+ x16 x26 x36 x46 x56 x66) 3))
(assert (or (= (+ x11 x21 x31) 1) (= (+ x11 x21 x31) 2)))
...
(assert (or (= (+ x64 x65 x66) 1) (= (+ x64 x65 x66) 2)))
(assert (or (not (= x11 x21)) (not (= x12 x22)) (not (= x13 x23))
            (not (= x14 x24)) (not (= x15 x25)) (not (= x16 x26))))
...
(assert (or (not (= x15 x16)) (not (= x25 x26)) (not (= x35 x36))
            (not (= x45 x46)) (not (= x55 x56)) (not (= x65 x66))))
(assert (= x22 0))   (assert (= x45 0))  ...  (assert (= x16 0))
(check-sat)
(get-model)
```

Propositional Logic in SMT-LIB 2

- `declare-const x Bool` creates propositional variable named `x`

Propositional Logic in SMT-LIB 2

- `declare-const x Bool` creates propositional variable named `x`
- `and or not implies` are used in prefix notation

Propositional Logic in SMT-LIB 2

- `declare-const x Bool` creates propositional variable named `x`
- `and or not implies` are used in prefix notation
- `assert` declares that formula must be satisfied

Propositional Logic in SMT-LIB 2

- `declare-const x Bool` creates propositional variable named `x`
- `and or not implies` are used in prefix notation
- `assert` declares that formula must be satisfied
- `check-sat` issues satisfiability test of conjunction of assertations

Propositional Logic in SMT-LIB 2

`declare-const x Bool`	creates propositional variable named `x`
`and or not implies`	are used in prefix notation
`assert`	declares that formula must be satisfied
`check-sat`	issues satisfiability test of conjunction of assertations
`get-model`	returns satisfying assignment (after satisfiability test)

Propositional Logic in SMT-LIB 2

- `declare-const x Bool` creates propositional variable named `x`
- `and or not implies` are used in prefix notation
- `assert` declares that formula must be satisfied
- `check-sat` issues satisfiability test of conjunction of assertions
- `get-model` returns satisfying assignment (after satisfiability test)

Links

- Z3

Propositional Logic in SMT-LIB 2

- `declare-const x Bool` creates propositional variable named `x`
- `and or not implies` are used in prefix notation
- `assert` declares that formula must be satisfied
- `check-sat` issues satisfiability test of conjunction of assertions
- `get-model` returns satisfying assignment (after satisfiability test)

Links

- Z3
- Z3 bindings for various programming languages

Propositional Logic in SMT-LIB 2

- `declare-const x Bool` creates propositional variable named `x`
- `and or not implies` are used in prefix notation
- `assert` declares that formula must be satisfied
- `check-sat` issues satisfiability test of conjunction of assertations
- `get-model` returns satisfying assignment (after satisfiability test)

Links

- Z3
- Z3 bindings for various programming languages
- Z3 bindings for Haskell

Propositional Logic in SMT-LIB 2

- `declare-const x Bool` creates propositional variable named `x`
- `and or not implies` are used in prefix notation
- `assert` declares that formula must be satisfied
- `check-sat` issues satisfiability test of conjunction of assertions
- `get-model` returns satisfying assignment (after satisfiability test)

Links

- Z3
- Z3 bindings for various programming languages
- Z3 bindings for Haskell
- SBV: SMT Based Verification in Haskell

Outline

SMT Problem

decide satisfiability of formulas in

propositional logic + domain-specific background theories

SMT Problem

decide satisfiability of formulas in

propositional logic + domain-specific background theories

Two Approaches

1. eager approach:

   translate formula into equisatisfiable propositional formula

SMT Problem

decide satisfiability of formulas in

propositional logic + domain-specific background theories

Two Approaches

1. eager approach:

   translate formula into equisatisfiable propositional formula

2. lazy approach:

   combine SAT solver with specialized solvers for background theories

SMT Problem

decide satisfiability of formulas in

propositional logic + domain-specific background theories

Two Approaches

1. eager approach:

   translate formula into equisatisfiable propositional formula

2. lazy approach:

   combine SAT solver with specialized solvers for background theories

Terminology

theory solver for T (T-solver) is procedure for deciding T-satisfiability of conjunction of quantifier-free literals

SMT Problem

decide satisfiability of formulas in

propositional logic + domain-specific background theories

Two Approaches

1. eager approach:

   translate formula into equisatisfiable propositional formula

2. lazy approach:

   combine SAT solver with specialized solvers for background theories

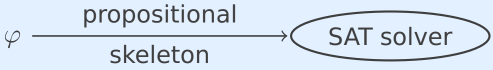
Terminology

theory solver for T (T-solver) is procedure for deciding T-satisfiability of conjunction of quantifier-free literals

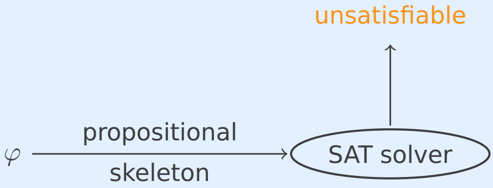
SMT Solving: Lazy Approach

φ

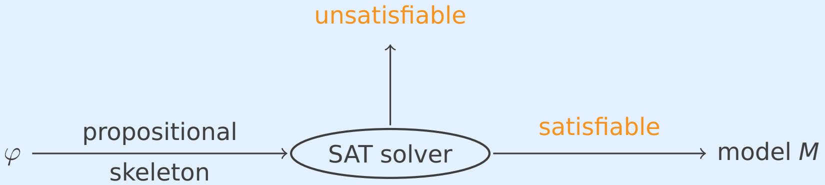
SMT Solving: Lazy Approach



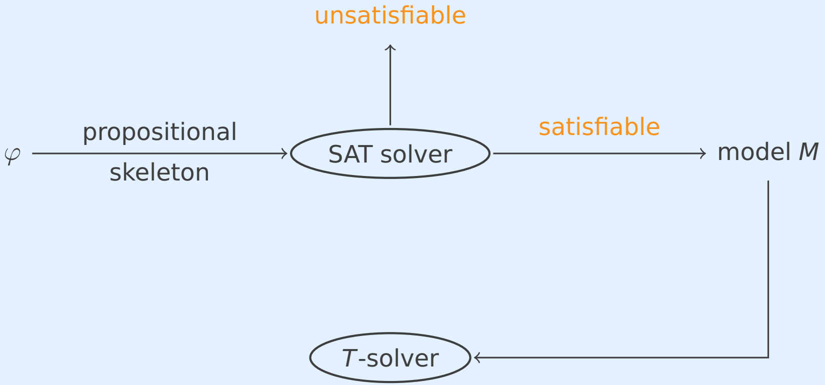
SMT Solving: Lazy Approach



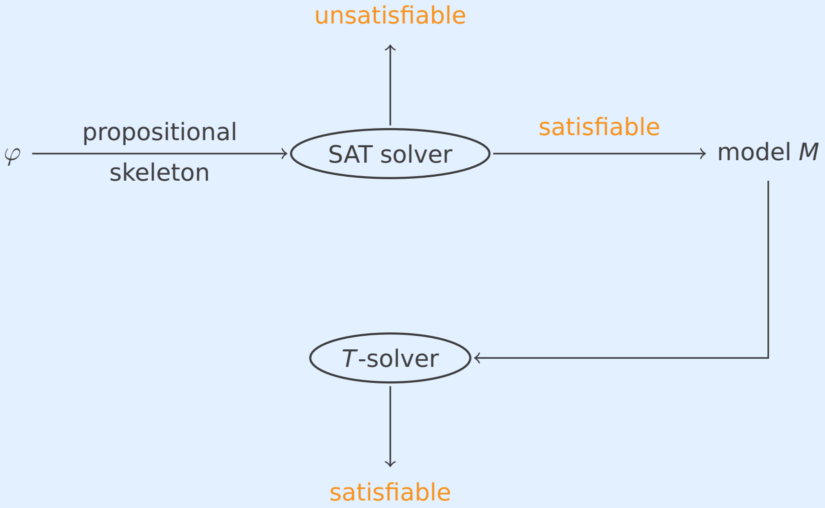
SMT Solving: Lazy Approach



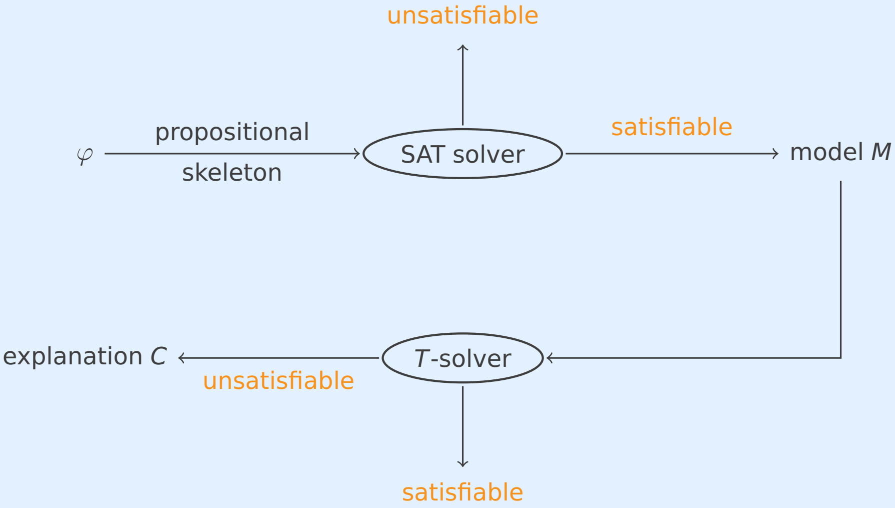
SMT Solving: Lazy Approach



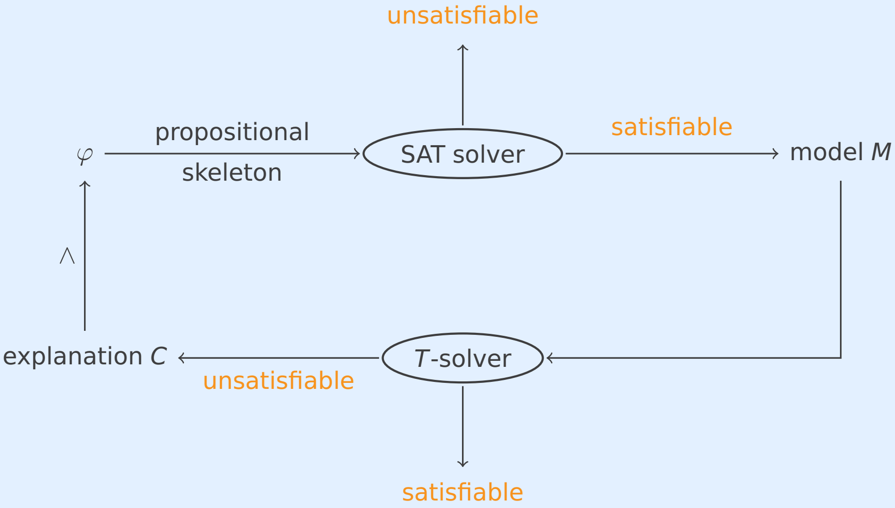
SMT Solving: Lazy Approach



SMT Solving: Lazy Approach



SMT Solving: Lazy Approach



Example

formula $x = 1 \wedge (\neg(y = 1) \vee \neg(x + 2y = 3)) \wedge x + y = 2$

Example

formula $\underbrace{x = 1}_{a} \wedge (\neg\underbrace{(y = 1)}_{b} \vee \neg\underbrace{(x + 2y = 3)}_{c}) \wedge \underbrace{x + y = 2}_{d}$

- input to SAT solver (propositional skeleton)

$$a \wedge (\neg b \vee \neg c) \wedge d$$

Example

formula $\underbrace{x = 1}_{a} \wedge (\neg\underbrace{(y = 1)}_{b} \vee \neg\underbrace{(x + 2y = 3)}_{c}) \wedge \underbrace{x + y = 2}_{d}$

- input to SAT solver (propositional skeleton)

$$a \wedge (\neg b \vee \neg c) \wedge d$$

- SAT solver reports satisfiable and returns model

$$a \wedge \neg b \wedge d$$

Example

formula $\underbrace{x = 1}_{a} \land (\neg\underbrace{(y = 1)}_{b} \lor \neg\underbrace{(x + 2y = 3)}_{c}) \land \underbrace{x + y = 2}_{d}$

- input to SAT solver (propositional skeleton)

$$a \land (\neg b \lor \neg c) \land d$$

- SAT solver reports satisfiable and returns model

$$a \land \neg b \land d$$

- input to LIA solver

$$x = 1 \land y \neq 1 \land x + y = 2$$

Example

formula $\underbrace{x = 1}_{a} \wedge (\neg(\underbrace{y = 1}_{b}) \vee \neg(\underbrace{x + 2y = 3}_{c})) \wedge \underbrace{x + y = 2}_{d}$

- input to SAT solver (propositional skeleton)

$$a \wedge (\neg b \vee \neg c) \wedge d$$

- SAT solver reports satisfiable and returns model

$$a \wedge \neg b \wedge d$$

- input to LIA solver

$$x = 1 \wedge y \neq 1 \wedge x + y = 2$$

- LIA solver reports unsatisfiable

Example

formula $\underbrace{x = 1}_{a} \wedge (\neg\underbrace{(y = 1)}_{b} \vee \neg\underbrace{(x + 2y = 3)}_{c}) \wedge \underbrace{x + y = 2}_{d}$

- input to SAT solver

$$a \wedge (\neg b \vee \neg c) \wedge d \wedge (\neg a \vee b \vee \neg d)$$

blocking clause

- SAT solver reports satisfiable and returns model

$$a \wedge \neg b \wedge d$$

- input to LIA solver

$$x = 1 \;\wedge\; y \neq 1 \;\wedge\; x + y = 2$$

- LIA solver reports unsatisfiable

Example

formula $\underbrace{x = 1}_{a} \wedge (\neg\underbrace{(y = 1)}_{b} \vee \neg\underbrace{(x + 2y = 3)}_{c}) \wedge \underbrace{x + y = 2}_{d}$

- input to SAT solver

$$a \wedge (\neg b \vee \neg c) \wedge d \wedge (\neg a \vee b \vee \neg d)$$

- SAT solver reports satisfiable and returns model

$$a \wedge b \wedge \neg c \wedge d$$

Example

formula $\underbrace{x = 1}_{a} \land (\neg\underbrace{(y = 1)}_{b} \lor \neg\underbrace{(x + 2y = 3)}_{c}) \land \underbrace{x + y = 2}_{d}$

- input to SAT solver

$$a \land (\neg b \lor \neg c) \land d \land (\neg a \lor b \lor \neg d)$$

- SAT solver reports satisfiable and returns model

$$a \land b \land \neg c \land d$$

- input to LIA solver

$$x = 1 \;\land\; y = 1 \;\land\; x + 2y \neq 3 \;\land\; x + y = 2$$

Example

formula $\underbrace{x = 1}_{a} \wedge (\neg\underbrace{(y = 1)}_{b} \vee \neg\underbrace{(x + 2y = 3)}_{c}) \wedge \underbrace{x + y = 2}_{d}$

- input to SAT solver

$$a \wedge (\neg b \vee \neg c) \wedge d \wedge (\neg a \vee b \vee \neg d)$$

- SAT solver reports satisfiable and returns model

$$a \wedge b \wedge \neg c \wedge d$$

- input to LIA solver

$$x = 1 \ \wedge \ y = 1 \ \wedge \ x + 2y \neq 3 \ \wedge \ x + y = 2$$

- LIA solver reports unsatisfiable

Example

formula $\underbrace{x = 1}_{a} \;\wedge\; (\neg\underbrace{(y = 1)}_{b} \;\vee\; \neg\underbrace{(x + 2y = 3)}_{c}) \;\wedge\; \underbrace{x + y = 2}_{d}$

- input to SAT solver

$$a \wedge (\neg b \vee \neg c) \wedge d \wedge (\neg a \vee b \vee \neg d) \wedge (\neg a \vee \neg b \vee c)$$

- SAT solver reports satisfiable and returns model

$$a \wedge b \wedge \neg c \wedge d$$

- input to LIA solver

$$x = 1 \;\wedge\; y = 1 \;\wedge\; x + 2y \neq 3 \;\wedge\; x + y = 2$$

- LIA solver reports unsatisfiable

Example

formula $\underset{a}{x = 1} \;\wedge\; (\neg\underset{b}{(y = 1)} \;\vee\; \neg\underset{c}{(x + 2y = 3)}) \;\wedge\; \underset{d}{x + y = 2}$

- input to SAT solver

$$a \wedge (\neg b \vee \neg c) \wedge d \wedge (\neg a \vee b \vee \neg d) \wedge (\neg a \vee \neg b \vee c)$$

- SAT solver reports unsatisfiable

Example

formula $\underbrace{x = 1}_{a} \wedge (\neg\underbrace{(y = 1)}_{b} \vee \neg\underbrace{(x + 2y = 3)}_{c}) \wedge \underbrace{x + y = 2}_{d}$ is unsatisfiable

- input to SAT solver

$$a \wedge (\neg b \vee \neg c) \wedge d \wedge (\neg a \vee b \vee \neg d) \wedge (\neg a \vee \neg b \vee c)$$

- SAT solver reports unsatisfiable

Outline

most state-of-the-art SMT solvers use DPLL(T)

most state-of-the-art SMT solvers use DPLL(T)

general framework for lazy SMT solving with theory propagation

most state-of-the-art SMT solvers use DPLL(T)

general framework for lazy SMT solving with theory propagation

Definitions

first-order theory T, formulas F and G, list of literals M

- F is T-satisfiable if $F \wedge T$ is satisfiable

most state-of-the-art SMT solvers use DPLL(T)

general framework for lazy SMT solving with theory propagation

Definitions

first-order theory T, formulas F and G, list of literals M

- F is T-satisfiable if $F \wedge T$ is satisfiable
- $F \models_T G$ if $F \wedge \neg G$ is not T-satisfiable

most state-of-the-art SMT solvers use DPLL(T)

general framework for lazy SMT solving with theory propagation

Definitions

first-order theory T, formulas F and G, list of literals M

- F is T-satisfiable if $F \wedge T$ is satisfiable
- $F \vDash_T G$ if $F \wedge \neg G$ is not T-satisfiable
- $F \equiv_T G$ if $F \vDash_T G$ and $G \vDash_T F$

most state-of-the-art SMT solvers use DPLL(T)

general framework for lazy SMT solving with theory propagation

Definitions

first-order theory T, formulas F and G, list of literals M

- F is T-satisfiable if $F \wedge T$ is satisfiable
- $F \vDash_T G$ if $F \wedge \neg G$ is not T-satisfiable
- $F \equiv_T G$ if $F \vDash_T G$ and $G \vDash_T F$
- $M = l_1, \dots, l_k$ is T-consistent if $l_1 \wedge \cdots \wedge l_k$ is T-satisfiable

Definition

DPLL(T) consists of DPLL rules unit propagate, decide, fail, restart

Definition

DPLL(T) consists of DPLL rules unit propagate, decide, fail, restart and

- T-backjump $\qquad M \overset{d}{l} N \parallel F, C \implies M\, l' \parallel F, C$

  if $M \overset{d}{l} N \vDash \neg C$ and $\exists$ clause $C' \vee l'$ such that

  - $F, C \vDash_T C' \vee l'$ and $M \vDash \neg C'$
  - l' is undefined in M and l' or l'^c occurs in F or in $M \overset{d}{l} N$

Definition

DPLL(T) consists of DPLL rules unit propagate, decide, fail, restart and

- T-backjump $$M \overset{d}{l} N \parallel F, C \quad \Longrightarrow \quad M l' \parallel F, C$$

  if $M \overset{d}{l} N \vDash \neg C$ and $\exists$ clause $C' \vee l'$ such that

  - $F, C \vDash_T C' \vee l'$ and $M \vDash \neg C'$
  - l' is undefined in M and l' or l'^c occurs in F or in $M \overset{d}{l} N$

- T-learn $$M \parallel F \quad \Longrightarrow \quad M \parallel F, C$$

  if $F \vDash_T C$ and all atoms of C occur in M or F

Definition

DPLL(T) consists of DPLL rules unit propagate, decide, fail, restart and

- T-backjump $$M \; \overset{d}{l} \; N \parallel F, C \quad \Longrightarrow \quad M \; l' \parallel F, C$$

  if $M \; \overset{d}{l} \; N \vDash \neg C$ and $\exists$ clause $C' \vee l'$ such that

  - $F, C \vDash_T C' \vee l'$ and $M \vDash \neg C'$
  - l' is undefined in M and l' or l'^c occurs in F or in $M \; \overset{d}{l} \; N$

- T-learn $$M \parallel F \quad \Longrightarrow \quad M \parallel F, C$$

  if $F \vDash_T C$ and all atoms of C occur in M or F

- T-forget $$M \parallel F, C \quad \Longrightarrow \quad M \parallel F$$

  if $F \vDash_T C$

Definition

DPLL(T) consists of DPLL rules unit propagate, decide, fail, restart and

- T-backjump $\qquad M\ l^d\ N \parallel F, C \implies M\ l' \parallel F, C$

  if $M\ l^d\ N \vDash \neg C$ and $\exists$ clause $C' \vee l'$ such that

  - $F, C \vDash_T C' \vee l'$ and $M \vDash \neg C'$
  - l' is undefined in M and l' or l'^c occurs in F or in $M\ l^d\ N$

- T-learn $\qquad M \parallel F \implies M \parallel F, C$

  if $F \vDash_T C$ and all atoms of C occur in M or F

- T-forget $\qquad M \parallel F, C \implies M \parallel F$

  if $F \vDash_T C$

- T-propagate $\qquad M \parallel F \implies M\ l \parallel F$

  if $M \vDash_T l$, l is undefined in M, and l or l^c occurs in F

Example

(EUF) formula $\underbrace{g(a) = c}_{1} \wedge (\neg \underbrace{(f(g(a)) = f(c))}_{2} \vee \underbrace{g(a) = d}_{3}) \wedge \neg \underbrace{(c = d)}_{4}$

Example

(EUF) formula $\underbrace{g(a) = c}_{1} \wedge (\neg(\underbrace{f(g(a)) = f(c)}_{2}) \vee \underbrace{g(a) = d}_{3}) \wedge \neg(\underbrace{c = d}_{4})$

$\parallel\ 1,\ \neg 2 \vee 3,\ \neg 4$

Example

(EUF) formula $\underbrace{g(a) = c}_{1} \wedge (\neg\underbrace{(f(g(a)) = f(c))}_{2} \vee \underbrace{g(a) = d}_{3}) \wedge \neg\underbrace{(c = d)}_{4}$

$$\parallel\ 1,\ \neg 2 \vee 3,\ \neg 4$$

$$\Longrightarrow \quad 1 \ \parallel\ 1,\ \neg 2 \vee 3,\ \neg 4$$

unit propagate

Example

(EUF) formula $\underbrace{g(a) = c}_{1} \land (\neg \underbrace{(f(g(a)) = f(c))}_{2} \lor \underbrace{g(a) = d}_{3}) \land \neg \underbrace{(c = d)}_{4}$

$$\| \; 1, \; \neg 2 \lor 3, \; \neg 4$$

$$\Longrightarrow \quad 1 \; \| \; 1, \; \neg 2 \lor 3, \; \neg 4 \qquad \text{unit propagate}$$

$$\Longrightarrow \quad 1 \; \neg 4 \; \| \; 1, \; \neg 2 \lor 3, \; \neg 4 \qquad \text{unit propagate}$$

Example

(EUF) formula $\underbrace{g(a) = c}_{1} \wedge (\neg\underbrace{(f(g(a)) = f(c))}_{2} \vee \underbrace{g(a) = d}_{3}) \wedge \neg\underbrace{(c = d)}_{4}$

$$\begin{array}{lrll}
 & & \| \ 1,\ \neg 2 \vee 3,\ \neg 4 & \\
\Longrightarrow & 1 & \| \ 1,\ \neg 2 \vee 3,\ \neg 4 & \text{unit propagate} \\
\Longrightarrow & 1\ \neg 4 & \| \ 1,\ \neg 2 \vee 3,\ \neg 4 & \text{unit propagate} \\
\Longrightarrow & 1\ \neg 4\ \neg 2^{d} & \| \ 1,\ \neg 2 \vee 3,\ \neg 4 & \text{decide}
\end{array}$$

Example

(EUF) formula $\underbrace{g(a) = c}_{1} \wedge (\neg \underbrace{(f(g(a)) = f(c))}_{2} \vee \underbrace{g(a) = d}_{3}) \wedge \neg \underbrace{(c = d)}_{4}$

		$\parallel$	$1,\ \neg 2 \vee 3,\ \neg 4$	
$\Longrightarrow$	1	$\parallel$	$1,\ \neg 2 \vee 3,\ \neg 4$	unit propagate
$\Longrightarrow$	$1\ \neg 4$	$\parallel$	$1,\ \neg 2 \vee 3,\ \neg 4$	unit propagate
$\Longrightarrow$	$1\ \neg 4\ \neg 2^{d}$	$\parallel$	$1,\ \neg 2 \vee 3,\ \neg 4$	decide
$\Longrightarrow$	$1\ \neg 4\ \neg 2^{d}$	$\parallel$	$1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4$	T-learn

Example

(EUF) formula $\underset{1}{g(a) = c} \wedge (\neg(\underset{2}{f(g(a)) = f(c)}) \vee \underset{3}{g(a) = d}) \wedge \neg(\underset{4}{c = d})$

		$\parallel$	$1,\ \neg 2 \vee 3,\ \neg 4$	
$\Longrightarrow$	1	$\parallel$	$1,\ \neg 2 \vee 3,\ \neg 4$	unit propagate
$\Longrightarrow$	$1\ \neg 4$	$\parallel$	$1,\ \neg 2 \vee 3,\ \neg 4$	unit propagate
$\Longrightarrow$	$1\ \neg 4\ \neg 2^d$	$\parallel$	$1,\ \neg 2 \vee 3,\ \neg 4$	decide
$\Longrightarrow$	$1\ \neg 4\ \neg 2^d$	$\parallel$	$1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4$	T-learn
$\Longrightarrow$	$1\ \neg 4\ 2$	$\parallel$	$1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4$	T-backjump

Example

(EUF) formula $\underbrace{g(a) = c}_{1} \wedge (\neg(\underbrace{f(g(a)) = f(c)}_{2}) \vee \underbrace{g(a) = d}_{3}) \wedge \neg(\underbrace{c = d}_{4})$

		$\Vert$	$1,\ \neg 2 \vee 3,\ \neg 4$	
$\Longrightarrow$	1	$\Vert$	$1,\ \neg 2 \vee 3,\ \neg 4$	unit propagate
$\Longrightarrow$	$1\ \neg 4$	$\Vert$	$1,\ \neg 2 \vee 3,\ \neg 4$	unit propagate
$\Longrightarrow$	$1\ \neg 4\ \neg 2^{d}$	$\Vert$	$1,\ \neg 2 \vee 3,\ \neg 4$	decide
$\Longrightarrow$	$1\ \neg 4\ \neg 2^{d}$	$\Vert$	$1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4$	T-learn
$\Longrightarrow$	$1\ \neg 4\ 2$	$\Vert$	$1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4$	T-backjump
$\Longrightarrow$	$1\ \neg 4\ 2\ 3$	$\Vert$	$1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4$	unit propagate

Example

(EUF) formula $\underbrace{g(a) = c}_{1} \wedge (\neg(\underbrace{f(g(a)) = f(c)}_{2}) \vee \underbrace{g(a) = d}_{3}) \wedge \neg(\underbrace{c = d}_{4})$

		$\parallel$	$1,\ \neg 2 \vee 3,\ \neg 4$	
$\Longrightarrow$	1	$\parallel$	$1,\ \neg 2 \vee 3,\ \neg 4$	unit propagate
$\Longrightarrow$	$1\ \neg 4$	$\parallel$	$1,\ \neg 2 \vee 3,\ \neg 4$	unit propagate
$\Longrightarrow$	$1\ \neg 4\ \neg 2^{d}$	$\parallel$	$1,\ \neg 2 \vee 3,\ \neg 4$	decide
$\Longrightarrow$	$1\ \neg 4\ \neg 2^{d}$	$\parallel$	$1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4$	T-learn
$\Longrightarrow$	$1\ \neg 4\ 2$	$\parallel$	$1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4$	T-backjump
$\Longrightarrow$	$1\ \neg 4\ 2\ 3$	$\parallel$	$1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4$	unit propagate
$\Longrightarrow$	$1\ \neg 4\ 2\ 3$	$\parallel$	$1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4,\ \neg 1 \vee \neg 2 \vee \neg 3 \vee 4$	T-learn

Example

(EUF) formula $\underbrace{g(a) = c}_{1} \wedge (\neg(\underbrace{f(g(a)) = f(c)}_{2}) \vee \underbrace{g(a) = d}_{3}) \wedge \neg(\underbrace{c = d}_{4})$

$$
\begin{array}{llll}
 & & \| \; 1,\ \neg 2 \vee 3,\ \neg 4 & \\
\Longrightarrow & 1 & \| \; 1,\ \neg 2 \vee 3,\ \neg 4 & \text{unit propagate} \\
\Longrightarrow & 1\ \neg 4 & \| \; 1,\ \neg 2 \vee 3,\ \neg 4 & \text{unit propagate} \\
\Longrightarrow & 1\ \neg 4\ \neg 2^{d} & \| \; 1,\ \neg 2 \vee 3,\ \neg 4 & \text{decide} \\
\Longrightarrow & 1\ \neg 4\ \neg 2^{d} & \| \; 1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4 & T\text{-learn} \\
\Longrightarrow & 1\ \neg 4\ 2 & \| \; 1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4 & T\text{-backjump} \\
\Longrightarrow & 1\ \neg 4\ 2\ 3 & \| \; 1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4 & \text{unit propagate} \\
\Longrightarrow & 1\ \neg 4\ 2\ 3 & \| \; 1,\ \neg 2 \vee 3,\ \neg 4,\ \neg 1 \vee 2 \vee 4,\ \neg 1 \vee \neg 2 \vee \neg 3 \vee 4 & T\text{-learn} \\
\Longrightarrow & \text{fail-state} & & \text{fail}
\end{array}
$$

Remark

lazy SMT approach is modeled in DPLL(T)

Remark

lazy SMT approach is modeled in DPLL(T) as follows:

if state $M \parallel F$ is reached such that unit propagate, decide, fail, T-backjump are not applicable

Remark

lazy SMT approach is modeled in DPLL(T) as follows:

if state $M \parallel F$ is reached such that unit propagate, decide, fail, T-backjump are not applicable

check T-consistency of M with T-solver

Remark

lazy SMT approach is modeled in DPLL(T) as follows:

if state $M \parallel F$ is reached such that unit propagate, decide, fail, T-backjump are not applicable

check T-consistency of M with T-solver

1. if M is T-consistent then F is T-satisfiable

Remark

lazy SMT approach is modeled in DPLL(T) as follows:

if state $M \parallel F$ is reached such that unit propagate, decide, fail, T-backjump are not applicable

check T-consistency of M with T-solver

1. if M is T-consistent then F is T-satisfiable
2. if M is not T-consistent then $F \vDash_T \neg(l_1 \wedge \cdots \wedge l_k)$ for some literals $l_1, \ldots, l_k$ in M

Remark

lazy SMT approach is modeled in DPLL(T) as follows:

if state $M \parallel F$ is reached such that unit propagate, decide, fail, T-backjump are not applicable

check T-consistency of M with T-solver

1. if M is T-consistent then F is T-satisfiable
2. if M is not T-consistent then $F \vDash_T \neg(l_1 \wedge \cdots \wedge l_k)$ for some literals $l_1, \ldots, l_k$ in M

   add blocking clause $\neg l_1 \vee \cdots \vee \neg l_k$ by T-learn

Remark

lazy SMT approach is modeled in DPLL(T) as follows:

if state $M \parallel F$ is reached such that unit propagate, decide, fail, T-backjump are not applicable

check T-consistency of M with T-solver

1. if M is T-consistent then F is T-satisfiable
2. if M is not T-consistent then $F \vDash_T \neg(l_1 \wedge \cdots \wedge l_k)$ for some literals $l_1, \ldots, l_k$ in M

   add blocking clause $\neg l_1 \vee \cdots \vee \neg l_k$ by T-learn and apply restart

Remark

lazy SMT approach is modeled in DPLL(T) as follows:

if state $M \parallel F$ is reached such that unit propagate, decide, fail, T-backjump are not applicable

check T-consistency of M with T-solver

1. if M is T-consistent then F is T-satisfiable
2. if M is not T-consistent then $F \vDash_T \neg(l_1 \wedge \cdots \wedge l_k)$ for some literals $l_1, \ldots, l_k$ in M

   add blocking clause $\neg l_1 \vee \cdots \vee \neg l_k$ by T-learn and apply restart

Improvements

1. apply fail or T-backjump after T-learn (instead of restart)

Remark

lazy SMT approach is modeled in DPLL(T) as follows:

if state $M \parallel F$ is reached such that unit propagate, decide, fail, T-backjump are not applicable

check T-consistency of M with T-solver

1. if M is T-consistent then F is T-satisfiable
2. if M is not T-consistent then $F \models_T \neg(l_1 \wedge \cdots \wedge l_k)$ for some literals $l_1, \ldots, l_k$ in M

   add blocking clause $\neg l_1 \vee \cdots \vee \neg l_k$ by T-learn and apply restart

Improvements

1. apply fail or T-backjump after T-learn (instead of restart)
2. check T-consistency of M or apply T-propagate before decide

Remark

lazy SMT approach is modeled in DPLL(T) as follows:

if state $M \parallel F$ is reached such that unit propagate, decide, fail, T-backjump are not applicable

check T-consistency of M with T-solver

1. if M is T-consistent then F is T-satisfiable
2. if M is not T-consistent then $F \vDash_T \neg(l_1 \wedge \cdots \wedge l_k)$ for some literals $l_1, \ldots, l_k$ in M

   add blocking clause $\neg l_1 \vee \cdots \vee \neg l_k$ by T-learn and apply restart

Improvements

1. apply fail or T-backjump after T-learn (instead of restart)
2. check T-consistency of M or apply T-propagate before decide
3. find small unsatisfiable cores to minimize k in blocking clauses

Outline

Kröning and Strichmann

- Section 1.4
- Chapter 3

Kröning and Strichmann

- Section 1.4
- Chapter 3

Further Reading

- Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli
  Solving SAT and SAT Modulo Theories: From an Abstract Davis–Putnam–Logemann–Loveland Procedure to DPLL(T)
  Journal of the ACM 53(6), pp. 937 – 977, 2006

Important Concepts

- $\equiv_T$
- $\vDash_T$
- blocking clause
- complete theory
- conjunctive fragment
- consistent theory
- decidable theory
- DPLL(T)
- first-order formula
- fragment
- model
- Peano arithmetic
- propositional skeleton
- quantifier-free fragment
- sentence
- standard model
- T-backjump
- T-consistency
- T-learn
- T-propagate
- T-satisfiability
- T-solver